<u>CS320 Final Review</u>

- If set A has n finite elements in it then there are 2^n possible subset combinations
- A function from A->B is defined as f is a subset of AXB
- Range = the set of values actually assigned vs. Target = all possible outcomes
- Injective Function = every element in the domain maps to exactly one range in the target (1-1)
- The inverse of a function means that f(A)->A
- A function must be injective in order for an inverse function to exist
- **The empty set fi is a subset of every set**
- The cardinality of A is less than the cardinality of B if there exists injection from A to B
- For the cardinality of A to equal the cardinality of B need injection from A to B and injection from B to A
- Infinite Set = contains a proper subset of the same cardinality
- Aleph null has the same cardinality as the set of natural numbers, and it is the first smallest infinite cardinality (but is countable).
- Goedel number needs to contain a sequence of sequential primes- can't be an odd number
- If a word has K letters in it, then there are k + 1 possible splits
- Total function is defined everywhere vs. partial function, only a subset of the domain is defined
- Regular Expression = a string over the alphabet {a, b, c, lamda, fi, (,), U, concatenation, *} (# of elements + 7)
- The class of regular languages over sigma is defined as: empty set, {lamda}, singleton sets
- Context Free grammar is regular if it follows the form: A -> a, A -> lamda, A -> aB

  Ex. S -> lamda | aS | bS   (aUb)*
- DFA- delta is total, NFA- delta is partial
- NFA = can have more than one way to go, can have lamda transitions, may be missing some transitions. Accepts a string that potentially leads to acceptance in one way but not in all ways

- If regular language then: (exists K>0)(for all w that exist in language L)(|w| greater than or equal to k) → ((exists x,y,z over sigma*)(w=xyz ^ |y|>0 ^ (|xy| less than or equal to k) ((for all I greater than or equal to 0)(xy^iz exists in L))) = just by the fact that it's a regular language, then exists the constant k, and for every word that is size k or greater, exists a non-empty substring within the first k letters that contains your pumping window
- Language doesn't pump = not regular

- A regular language is represented by a regular expression, a regular context free grammar, or a NFA or DFA
- Regular grammar produces regular language
- Context Free grammar is the grammar of matched parenthesis
- Every regular expression is context free
- If a grammar is not context free, then it does not generate regular languages
- If context free language then: (exists K>0)(for all w that exist in language L)(|w| greater than or equal to k) → ((exists x,y_1, t, y_2, z over sigma*)(w=xy_1ty_2z ^ |$y_1$$y_2$|>0 ^ (|$y_1ty_2$| less than or equal to k) ((for all I greater than or equal to 0)(xy^1zy^2 exists in L))) = just by the fact that it's a regular language, then exists the constant k, and for every word that is size k or greater, exists a non-empty substring within the first k letters that contains your pumping window
- Delta(q, lamda) is the terminal configuration, its accepting if q is a final state, otherwise its rejecting
- Non-Deterministic Push Down Automaton:
    - M=(Q, sigma, gamma, delta, q0, F)
    - [q, a, B//pop, p, D//push] //anything pushed gets popped in reverse order
    - Accepts when finish input, finish stack, reach final state
    - M accepts w if there exists at least one computation that takes M from the initial computation to an accepting computation- if certain transitions fire at the wrong time it's ok, as long as it won't work but at least it won't create bad strings
- Class of context free languages is not closed under intersection
- If you can do compliment than you can do intersection and vice versa
- Deterministic Turing Machine:
    - M=(Q, sigma, gamma, delta, q0)
    - [q//current state, a//current symbol, p//next state, b//over-write symbol, D]
    - M halts when the transition for a given state and symbol doesn't exist in delta
    - If a turring machine accepts language L then it will accept all words w in L by halting: w exists in L -> M(w) //m halts on w, else m diverges on w
- Non-Deterministic Turing Machine = has more than one way to get to a given transition
- Turing Machine:
    - M=(Q, sigma, gamma, delta, q0, F)
    - Accepts a language by halting in final state
    - Else diverges (never halts), or rejects by halting not in final state

- Recursively Enumerable Language = exists a Turing Machine that accepts L
- A turing machine **enumerates** a language if it generates all words in the language
- Decidable Language = exists a turing machine that accepts L and halts on every input (accepts or rejects)
- A language is decidable if it has a tm that enumerates it in shortflex order
- Every regular language is decidable //because can create a turing machine that will accept all its input by escaping until hits a blank and not know what that is (see algorithm below)
- Every Context Free Language is decidable
- Universal Turing Machine is a machine that takes in a machine as input and reacts based on the outcome of its argument machine
- Universal Turing Machine simulates both deterministic and non-deterministic turing machines. If there are infinite tupples to check then simulate a finite amount of tupples via a counter
- $M_h(M,w)$ is defined as follows: halt and accept if M(w) halts, halt and reject if M(w) diverges -> DNE!!!
- Rice's Theorum: $M_B(M)$ is defined as follows: halts and accepts if B(L(M))= 1, halts and rejects if B(L(M))= 0 -> DNE!! (Decides a non-trivial property of the language that Machine m accepts): a turing machine cannot exist that decides the set of Turing machines whose languages satisfy any non-trivial property
- ****If L and L compliment are recursively enumerable then they are both also decidable**
- Thm: if M is a DFA that accepts L then L is non-empty iff M accepts at least one string of length less than or equal to k where k is the number of states in M
- Thm: if L is a regular language accepted by a deterministic finite automaton, then L is infinite iff F accepts a word of length greater than or equal to k but less than 2k
- **We can determine a lot of things with turing machines, but with grammars we can only determine if the language generated by the grammar is empty or not**
- Shorflex order = |x| less than |y| else (if =) refer to dictionary order- for every string there are finitely many strings that precede it in shortflex order
- ****If two languages are recursively enumerable, then their complement is not recursively enumerable**
- L1 (decidable), L2 (recursively enumerable): L1-L2 is not recursively enumerable

- Universal Turing Machine = can simulate the code of any other machine by:
  - Input: a) description of the machine we want to simulate b) input we want to simulate the machine with
  - Goal: Simulate M's execution when given input w
  - Result: will loop, reject, or accept just like M would- recursively enumerable, but not decidable
- Halting Problem = is the program looping forever or is it just taking a long time to run? Can't know unless it halts. If it's not going to halt, won't know.

Problem Solving

- For regexes with a specified range (ex. more than 3 but less than 6), use lamda as one of your options once you've reached the min quota
- Pumping Lemma:
  i. Adversary chooses k: Let k>0
  ii. You choose the pumping word: Select n greater than or equal to k
  iii. (State a property)
  iv. Adversary chooses a pumping decomposition: w=xyz
  v. Find i such that xy^iz not in L
- Context free intersect regular is context free: if can't prove via pumping lemma try via intersection
- Reducibility:
  - fact- whether or not a TM as input will accept its string as input is undecidable but recognizable
  - assume you do have a decider; then use that decider to decide something bigger that we know is undecidable

Algorithm

1. Regular Expression -> Grammar
   a. Base case: S -> lambda, S -> a, fi- no rule
   b. If the regular expression has operators then use algorithm 1

2. G1 + G2
   a. U = {S -> S1 | S2}
   b. Concatenation = {S -> S1S2}
   c. * = {S-> lambda | SS | S1}

3. Regular Expression -> NFA (can't have any incoming our outgoing arcs)
    a. Base case: lamda, singleton, fi – draw their automata
    b. If there are operators, combine union, concat, and * automata combos

4. Non-Deterministic Finite Automata -> Deterministic Finite Automata
    a. Make a transition state grid and include a column for C(x)
    b. Create a new transition state grid with C(x) as the new states (fi is a state)
    c. Draw the new Deterministic Finite Automata
    d. The new final state is any state that contains the old final state

5. Regular Grammar -> Automata (needs to have a single final state with no out degrees, and no lamda arcs except to the final state)
    a. q0 = S
    b. F = {Z} //need one more state that isn't a variable of the grammar
    c. Convert to proper form
    d. Combine the arcs and states (ex. A -> aB, B -> lamda)

6. Automata -> Regex (no in or out degrees, need on final state)
    a. Use GEG (generalized expression graph) to eliminate nodes
    b. Create state transition grids using the regular expressions as the transitions
7. Finite Autamaton -> Push Down Autamaton
    a. [q, a, p] -> [q, a, lamda, p, lamda]

    ****reverse algorithm doesn't exist**

8. **P.D.A + F.A**
    a. **[p, a, A, t, B] + [s, a, v] = [(p,s), a, A, (t,v), B]**

9. Deterministic Turing Machine -> Turing Machine

**if L is accepted by halting in a Deterministic Turing Machine, then there exists a Turing Machine that accepts L by final state

    a. Make every state a final state

10. Turing Machine -> Deterministic Turing Machine
    a. If M diverges make M’ diverge
    b. If M halts make M’ halt
    c. If M rejects, send M’ into infinite escape

11. Deterministic Finite Automaton -> Turing Machine that decides L(M)
    a. [p, a, t] -> [p, a, t, a, R] //a turing machine that doesn’t write anything and only goes to the right is the same thing a
    b. Will eventually halt since the delta of dfa doesn’t contain blank transitions so the new tm won’t know what to do with them either and will therefore halt
    **reverse algorithm doesn’t exist
12. CFG -> is L(G) empty
    a. Systematically go through all the variables and determine if they have a terminal
    b. If they do mark it off
    c. If S gets marked then L(G) is not empty

Counting

- L = number of choices in the regex times each other
- **(Lambda)(fi)(a) = 0**
- **Lamda U fi U a = 1**
- **(lamda*)(fi*)(a*) = aleph null**
- **Fi* = 1 (since it has lamda)**

Closures

L1	L2	L1L2	L1*	L1 (bar)	L1 intersect L2
CF	**CF**	CF	CF	Dec	Dec
Rec.enum	**Rec.enum**	Rec.enum	Rec.enum	**Not-rec.enum**	Rec.enum
Dec	**Dec**	Dec	Dec	Dec	Dec
CF	**Dec**	CF	CF	Dec	CF